

9-25-00

JC839 U.S. PTO

002260 9/22/00

Please type a plus sign (+) inside this box → +

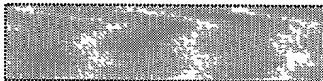
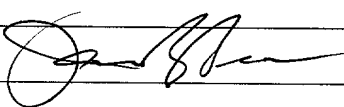
PTO/SB/05 (4/98)

Approved for use through 09/03/00. OMB 0651-0032

Patent and Trademark Office. U S DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

+

<b>UTILITY</b> <b>PATENT APPLICATION</b> <b>TRANSMITTAL</b> <small>(Only for new nonprovisional applications under 37 CFR 1.53(b))</small>		Attorney Docket No. 16356.549 (DC-02461)	
		First Named Inventor or Application Identifier Josef Zeevi et al.	
		Express Mail Label No. <b>EE 308375550US</b>	
APPLICATION ELEMENTS See MPEP chapter 600 concerning utility patent application contents		ADDRESS TO: Commissioner for Patents Box Patent Application Washington, DC 20231	
1. <input checked="" type="checkbox"/> Fee Transmittal Form <small>(Submit an original, and a duplicate for fee processing)</small> 2. <input checked="" type="checkbox"/> Specification [Total Pages <u>22</u> ] <small>(preferred arrangement set forth below)</small> <ul style="list-style-type: none"> <li>- Descriptive title of the Invention</li> <li>- Cross References to Related Applications</li> <li>- Statement Regarding Fed sponsored R &amp; D</li> <li>- Reference to Microfiche Appendix</li> <li>- Background of the Invention</li> <li>- Brief Summary of the Invention</li> <li>- Brief Description of the Drawings (if filed)</li> <li>- Detailed Description</li> <li>- Claim(s)</li> <li>- Abstract of the Disclosure</li> </ul> 3. <input checked="" type="checkbox"/> Drawing(s) (35 USC 113) [Total Sheets <u>9</u> ] 4. Oath or Declaration [Total Pages <u>2</u> ] <ul style="list-style-type: none"> <li>a. <input checked="" type="checkbox"/> Newly executed (original or copy)</li> <li>b. <input type="checkbox"/> Copy from a prior application (37 CFR 1.63(d)) (for continuation/divisional with Box 17 completed)  <small>[Note Box 5 below]</small></li> <li>i. <input type="checkbox"/> <u>DELETION OF INVENTOR(S)</u>            Signed statement attached deleting Inventor(s) named in the prior application, see 37 CFR 1.63(d) and 1.33 (b).</li> </ul> 5. <input type="checkbox"/> Incorporation By Reference (usable if Box 4b is checked) The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being a part of the disclosure of the accompanying application and is hereby incorporated by reference therein.		6. <input type="checkbox"/> Microfiche Computer Program (Appendix) 7. <input type="checkbox"/> Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary) <ul style="list-style-type: none"> <li>a. <input type="checkbox"/> Computer Readable Copy</li> <li>b. <input type="checkbox"/> Paper Copy (identical to computer copy)</li> <li>c. <input type="checkbox"/> Statement verifying identity of above copies</li> </ul>	
		<b>ACCOMPANYING APPLICATION PARTS</b>	
		8. <input checked="" type="checkbox"/> Assignment Papers (cover sheet & document(s)) 9. <input type="checkbox"/> 37 CFR 3.73(b) Statement <input type="checkbox"/> Power of Attorney <small>(when there is an assignee)</small> 10. <input type="checkbox"/> English Translation Document (if applicable) 11. <input checked="" type="checkbox"/> Information Disclosure <input checked="" type="checkbox"/> Copies of IDS Statement (IDS)/PTO-1449 Citations 12. <input type="checkbox"/> Preliminary Amendment 13. <input checked="" type="checkbox"/> Return Receipt Postcard (MPEM 503) <small>(Should be specifically itemized)</small> 14. <input type="checkbox"/> Small Entity <input type="checkbox"/> Statement filed in prior application Statement(s) Status still proper and desired 15. <input type="checkbox"/> Certified Copy of Priority Document(s) <small>(if foreign Priority is claimed)</small> 16. <input checked="" type="checkbox"/> Other: <u>Express Mail Certificate</u> ;	
17. If a CONTINUING APPLICATION, check appropriate box and supply the requisite information: <input type="checkbox"/> Continuation <input type="checkbox"/> Divisional <input type="checkbox"/> Continuation-in-part (CIP) of prior application No: _____, filed: _____ prior application information: Examiner _____ Group/Art Unit _____			
<b>18. CORRESPONDENCE ADDRESS</b>			
<input type="checkbox"/> Customer Number or Bar Code Label  or <input checked="" type="checkbox"/> Correspondence address below			
NAME		David L. McCombs	
		Haynes and Boone, LLP	
ADDRESS		901 Main Street	
		Suite 3100	
CITY	Dallas	STATE	TX
		ZIP CODE	75202
COUNTRY	USA	TELEPHONE	214-651-5533
		FAX	214-651-5940
Name (print/type)	James R. Bell	Registration No. (Attorney/Agent)	26, 528
Signature		Date	9-22-00
a-103489			

JC542 U.S. PTO

09/666316

09/22/00

+

08-25-00

A

09/22/00  
jc839 U.S. PTO

PATENT

Docket Number: 16356.549 (DC-02461)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:  
Josef Zeevi et al.

Serial No. N/A

Filed: Herewith

For: A DIAGNOSTIC ARCHITECTURE FOR  
USE WITH AN INTERFACE BETWEEN  
AN OPERATING SYSTEM AND  
PLATFORM FIRMWARE

§  
§  
§  
§  
§  
§  
§  
§  
§  
§  
§

Examiner: N/A

Group Art Unit: N/A

jc542 U.S. PTO  
09/668316  
09/22/00

Box Patent Application  
Commissioner for Patents  
Washington, D.C. 20231

EXPRESS MAIL CERTIFICATE

Express Mail Number: EE308375550US

Date of Deposit: 9/22/00

I hereby certify that the following attached papers and fee:

1. Utility Patent Application Transmittal Sheet;
2. Fee Transmittal Sheet (in duplicate);
3. Patent Application consisting of 16 pages of Specification, 5 pages of Claims and 1 page Abstract of Disclosure;
4. 9 Drawing sheets;
5. Declaration and Power of Attorney;
6. Assignment and Recordation Cover Sheet;
7. Information Disclosure Statement with cited references;
8. Check in the amount of \$766.00 (\$726.00 for filing fee and \$40.00 for recordation of Assignment); and
9. A Return Postcard

are being deposited with United States Postal Service "Express Mail Post Office to Addressee" to the Commissioner for Patents, Box Patent Application, Washington, D.C. 20231.

Dolly Milleson  
Typed or Printed Name

Dolly Milleson  
Signature  
A-103491.1

09668316 092200

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

**FEE TRANSMITTAL  
for FY 2000**

Patent fees are subject to annual revision  
Small Entity payments must be supported by a small entity statement,  
otherwise large entity fees must be paid. See Forms PTO/SB/09-12  
See 37 C.F.R. §§ 1.27 and 1.28

**TOTAL AMOUNT OF  
PAYMENT** (\$ 766.00
**Complete if Known**

Application Number	N/A
Filing Date	Herewith
First Named Inventor	Josef Zeevi et al.
Group Art Unit	Unknown
Examiner Name	Unknown
Attorney Docket Number	16356.549 (DC-02461)

**METHOD OF PAYMENT (check one)**

1. ☒ The Commissioner is hereby authorized to charge indicated fees and credit and over payment to  
Deposit Account Number **08-1394**  
Deposit Account Name **Haynes and Boone**  
☐ Charge Any Additional Fee Required Under 37 CFR 1.16 and 1.17 ☐ Charge the Issue Fee in 37 CFR 1.18 at the Mailing of the Notice of Allowance
2. ☒ Payment Enclosed  
☒ Check ☐ Money Order ☐ Other

**FEE CALCULATION****1. FILING FEE**

Large Fee Code	Entity Fee (\$)	Small Fee Code	Entity Fee (\$)	Fee Description	Fee Paid
101	690	201	345	Utility filing fee	690
106	310	206	155	Design filing fee	
107	480	207	240	Plant filing fee	
108	690	208	345	Reissue filing fee	
114	150	214	75	Provisional filing fee	

**SUBTOTAL (1)** (\$ 690.00)**2. CLAIMS**

Total Claims	Extra	Fee	Fee Paid
22	-20 = 2	X 18	= 36.00
Independent Claims	3	-3 = 0	X 78 = 0.00
Multiple Dependent Claims	0	X 260	= 0.00

Large Fee Code	Entity Fee (\$)	Small Fee Code	Entity Fee (\$)	Fee Description
103	18	303	9	Claims in excess of 20
102	78	202	39	Independent claims in excess of 3
104	260	204	130	Multiple dependent claim
109	78	209	39	Reissue independent claims over original patent
110	18	210	9	Reissue claims in excess of 20 and over original patent

**SUBTOTAL (2)** (\$ 36.00)**FEE CALCULATION (continued)****410. ADDITIONAL FEES**

Large Fee Code	Entity Fee (\$)	Small Fee Code	Entity Fee (\$)	Fee Description	Fee Paid
105	130	205	65	Surcharge - late filing fee or oath	
127	50	227	25	Surcharge - late provisional filing fee or cover sheet	
139	130	139	130	Non-English specification	
147	2520	147	2520	For filing a request for reexamination	
112	920*	112	920*	Requesting publication of SIR prior to Examiner action	
113	1840*	113	1840*	Request publication of SIR after Examiner action	
115	110	215	55	Extension for reply within first month	
116	380	216	190	Extension for reply within second month	
117	870	217	435	Extension for reply within third month	
118	1360	218	680	Extension for reply within fourth month	
128	1850	228	925	Extension for reply within fifth month	
119	300	219	150	Notice of Appeal	
120	300	220	150	Filing a Brief in support of an appeal	
121	260	221	130	Request for oral hearing	
138	1510	138	1510	Petition to institute a public use proceeding	
140	110	240	55	Petition to revive - unavoidable	
141	1210	241	605	Petition to revive- unintentional	
142	1210	242	605	Utility issue fee (or reissue)	
143	430	243	215	Design issue fee	
144	580	244	290	Plant issue fee	
122	130	122	130	Petitions to the Commissioner	
123	50	123	50	Petitions related to provisional applications	
126	240	126	240	Submission of Information Disclosure Statement	
581	40	581	40	Recording each patent assignment per property (times number of properties)	40.00
146	690	246	345	Filing a submission after final rejection (37 CFR 1.129(a))	
149	690	249	345	For each additional invention to be examined (37 CFR 1.129(b))	

Other fee (specify) \_\_\_\_\_

Other fee (specify) \_\_\_\_\_

\* Reduced by Basic Filing Fee Paid

**SUBTOTAL (3)** (\$ 40.00)**SUBMITTED BY**Typed or Printed Name **James R. Bell**Signature Date **9-22-00****Complete (if applicable)**Reg. No. **26,528**Deposit Account **08-1394**  
User ID**WARNING:**

Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of Time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Box Patent Application, Washington, DC 20231

A-103488 1

09/22/00

09/22/00 09:22:00



EXPRESS MAIL NO.: <u>EE30837555045</u>	DATE OF DEPOSIT <u>9/22/00</u>
This paper and fee are being deposited with the U.S. Postal Service Express Mail Post Office to Addressee service under 37 CFR §1.10 on the date indicated above and is addressed to: Box PATENT APPLICATION, Commissioner for Patents, Washington, D.C. 20231	
<u>Dohy Milleson</u>	<u>Dohy Milleson</u>
Name of person mailing paper and fee	Signature of person mailing paper and fee

**A DIAGNOSTIC ARCHITECTURE FOR USE WITH AN INTERFACE BETWEEN  
AN OPERATING SYSTEM AND PLATFORM FIRMWARE**

Inventors: Josef Zeevi  
14801 Endicott Drive  
Austin, Texas 78728

Roderick Wayne Stone  
P.O. Box 181156  
Austin, Texas 78718

Assignee: Dell Products L.P.  
One Dell Way  
Round Rock, Texas 78682-2244

David L. McCombs  
HAYNES AND BOONE, L.L.P.  
901 Main Street  
Suite 3100  
Dallas, Texas 752023789  
(214) 651-5533

096634B 092200

**PATENT**

Docket No.: DC-02461 (16356.549)

EXPRESS MAIL NO.: <u>EE 30837555045</u>	DATE OF DEPOSIT <u>9/22/00</u>
This paper and fee are being deposited with the U.S. Postal Service Express Mail Post Office to Addressee service under 37 CFR §1.10 on the date indicated above and is addressed to: Box PATENT APPLICATION, Commissioner for Patents, Washington, D.C. 20231	
<u>Dolly Milleson</u>	<u>Dolly Milleson</u>
Name of person mailing paper and fee	Signature of person mailing paper and fee

**A DIAGNOSTIC ARCHITECTURE FOR USE WITH AN INTERFACE BETWEEN  
AN OPERATING SYSTEM AND PLATFORM FIRMWARE**

Background

5 The disclosures herein relate generally to computer systems and, more particularly, to a diagnostic architecture for use with an interface between an operating system and platform firmware.

10 This application relates to co-pending United States Patent Application Serial No. 09/333,786, filed on June 15, 1999, entitled "Method and Apparatus for Testing Custom-Configured Software/Hardware Integration in a Computer Build-to-Order Manufacturing Process", naming Thomas Vrhel Jr., Gaston M. Barajas, Paul J. Maia, and W.D. Todd Nix as inventors.

15 This application relates to co-pending United States Patent Application Serial No. 09/237,760, filed on January 26, 1999, entitled "A Method of Installing Software on and/or Testing a Computer System", naming Richard D. Amberg, Roger Wong, and Michael Lynch as inventors.

20 This application relates to co-pending United States Patent No. 5,995,757, filed on August 29, 1997, entitled "Software Installation and Testing for a Build-to-Order Computer System", naming Richard D. Amberg, Roger W. Wong, and Michael A. Brundridge as inventors.

This application relates to co-pending United States Patent No. 5,995,757, filed on August 29, 1997, entitled "Software Installation and Testing for a Build-to-Order Computer

**PATENT**

Docket No.: DC-02461 (16356.549)

System”, naming Richard D. Amberg, Roger W. Wong, and Michael A. Brundridge as inventors.

This application relates to co-pending United States Patent No. 5,963,743, filed on August 29, 1997, entitled “Database for Facilitating Software Installation and Testing for a Build-to-Order Computer System”, naming Richard D. Amberg, Roger W. Wong, and Michael A. Brundridge as inventors.

The co-pending applications are incorporated by reference in their entirety, and are assigned to the assignee of this application.

In the process of manufacturing a build-to-order computer system, diagnostic software may be copied to the computer system and executed to ensure the reliability of the system. Computer systems offered by a computer manufacturer, particularly build-to-order computer systems, may include different features and devices. Accordingly, the computer manufacturer may need to customize the diagnostic software for each platform that the manufacturer offers. The process of customizing diagnostic software for computer systems may require a costly and inefficient development effort.

In addition, diagnostic software may include one or more test modules associated with one or more devices provided by a third party vendor. As used herein, the term test module refers to a set of information that may be used to test or perform diagnostics or other services on a component of a computer system. The information may include, for example, one or more test routines, test information, device information, parameter information, and other programs. The test modules may be used to perform tests on the components of a computer system. The test modules provided by the third party vendors may not conform to a uniform standard or convention, however. As a result, additional development effort may be needed to overcome the lack of standardization and incorporate test modules provided by third party vendors into diagnostic software that may be used in a build-to-order manufacturing process.

Computer systems typically include an operating system and platform firmware and may include an interface between the operating system and the platform firmware. One such interface is described in the *Extensible Firmware Interface Specification* (hereafter "EFI"), version 0.99, published by Intel Corporation, 2200 Mission College Blvd., Santa Clara, CA 95052, (408) 765-8080, <http://www.intel.com>, Intel order # 731843-001, and incorporated by reference herein. Although EFI may be used in booting a computer system, it would be desirable to be able to use EFI in a diagnostic architecture.

Therefore, what is needed is a diagnostic architecture that allows a test module to be used in a build-to-order manufacturing environment. What is also needed is a diagnostic architecture that uses and enhances the capabilities of an interface between an operating system and platform firmware in a computer system.

#### Summary

One embodiment, accordingly, provides a method performed by a computer system. The method includes detecting a test module interface associated with a test module and calling a function identified by the test module interface to cause a test configuration of the test module to be created.

A principal advantage of this embodiment is that it allows a test module to be used in a build-to-order manufacturing environment. The test module is loaded onto a build-to-order computer system. A test module interface associated with the test module provides a user interface or other program with the ability to select test routines or other information in the test module. The test module interface may be created using the capabilities of an interface between an operating system and platform firmware.

#### Brief Description of the Drawings

Fig. 1 is a diagram illustrating an embodiment of computer system.

Fig. 2 is a diagram illustrating layers of software in a diagnostic architecture.

Fig. 3 is a diagram illustrating an example of the use of test module interfaces in a diagnostic architecture.

Fig. 4a is a diagram illustrating an embodiment of a registration table.

Fig. 4b is a first diagram illustrating an example of a registration table in operation.

Fig. 4c is a second diagram illustrating an example of a registration table in operation.

Fig. 4d is a third diagram illustrating an example of a registration table in operation.

Fig. 5a is a flowchart illustrating an embodiment of a method for using test module interfaces in a diagnostic architecture.

Fig. 5b is a flowchart illustrating an embodiment of a method for unloading a test module in a diagnostic architecture.

Fig. 5c is a flowchart illustrating an embodiment of a method for reinstalling a test module interface in a diagnostic architecture.

Fig. 6 is a diagram illustrating an example of a test configuration.

#### Detailed Description

Fig. 1 is a diagram illustrating an embodiment of a computer system 100. Computer system 100 is operable by a user 102 and includes a processor 110, a chipset 120, a memory 130, and devices 140a, 140b, and 140c as shown in block 100a. Device 140a is coupled to a port 142a in chipset 120. Devices 140b and 140c are coupled to a shared bus 142b in chipset 120. Computer system 100 may include other features not shown in Fig. 1.

In the process of manufacturing computer system 100, a computer manufacturer stores test modules 106 onto computer system 100 as indicated by arrow 108. As noted above, the term test module refers to a set of information that may be used to test or perform diagnostics or other services on a component of a computer system. The information may include, for example, one or more test routines, test information, device information, parameter information, and other programs. The test modules may be used to perform tests on the components of a computer system. Test modules 106 are configured to operate in a



diagnostic architecture described herein. In particular, each test module 106 is configured to expose a test module interface. A test module interface provides a user interface, an executive, or other program with the ability to access information and call functions within the test module. In the following description, an executive will be described as interacting with a test module interface. It is noted, however, that a user interface or other program may also interact with a test module interface in ways similar to those described below.

A test module interface provides an executive with the ability to detect information in a test module. In one particular embodiment, each test module interface provides a function called FCONFIGURE that is callable by an executive. Once called, FCONFIGURE generates a test configuration for the associated test module and returns a pointer to the test configuration to the executive. The test configuration includes information associated with the test module. This information may include information regarding one or more devices, information regarding one or more test functions, and information regarding one or more parameters. The test configuration may omit any of the above information or include other information as desired. Fig. 6, described below, illustrates an example of a test configuration.

Test modules 106 each cause an associated test module interface to be installed. After a test module 106 is loaded into a memory, such as memory 130 on computer system 100, a test module 106 causes its associated test module interface to be created and stored in the memory. A test module 106 may cause the test module interface to be created in response to certain conditions being met. These conditions may include the presence of a device associated with the test module 106, for example device 140a, device 140b, or device 140c, or the presence of information associated with the test module 106, for example another test module, another program, or data. If a condition is not met, the test module 106 does not install a test module interface and may unload from the memory.

An executive detects the installed test module interfaces within computer system 100. The executive causes a test configuration to be created for each test module 106 using an

associated test module interface as noted above. A test configuration of a test module 106 allows the user interface to detect test functions and other routines within the test module that may be executed by computer system 100. Other programs including other test modules may also detect test functions and other routines within a given test module 106 using its test configuration. The executive, in conjunction with a user interface, may cause a list of available test functions or routines to be displayed on a display of computer system 100. The executive is configured to call a test function or other routine for execution by computer system 100 and may do so in response to an input from user 102 received through a user interface.

By having test modules 106 conditionally install test module interfaces on computer system 100, a diagnostic architecture may be created for use in a build-to-order manufacturing environment. A computer manufacturer may install the same set of test modules 106 onto each computer system that it builds. Test modules 106 can each independently determine whether to install their test module interfaces in response to one or more conditions being met. If a condition for a particular test module 106 is not met, then that test module 106 does not install a test module interface. Accordingly, an executive would not cause a test configuration to be created for this test module 106 and would not cause test functions within the test module 106 to be executed. In contrast, an executive detects the installed test module interfaces. Using the installed test module interfaces, an executive can cause test configurations to be created and cause test functions to be executed. In this way, only the test modules 106 that are used by a particular computer system such as computer system 100 become available to the executive.

For example, a first test module 106 that is associated with device 140a may be loaded onto computer system 100. The first test module 106 detects the presence of device 140a and creates a test module interface. The executive detects the test module interface, causes a test configuration to be created, and causes test functions associated with device 140a to be executed on computer system 100. In this same example, a second test module

106 may also be loaded onto computer system 100. The second test module 106, however, is associated with a device or information that is not present on computer system 100.

Accordingly, the second test module 106 does not install its test module interface and may unload from the memory. As a result, the executive will not detect a test module interface associated with this second test module 106, and test programs within the second test module 106 will not be exposed to the executive. Although the presence or absence of device information was used as a condition for installing a test module interface in this example, other conditions may be used as noted above.

A test module may create a test module interface using features specified in the *Extensible Firmware Interface Specification* ("EFI") noted above and incorporated by reference. In particular, a test module interface is created as a protocol as defined by EFI. By creating a test module interface as an EFI protocol, other programs are given the ability to detect and interact with the test module interface. An executive, for example, detects each test module interface on computer system 100 by using EFI support for protocol registration. The executive and other programs such as a user interface also create an EFI protocol.

The diagnostic architecture described herein allows computer system 100 to automatically reconfigure without requiring a complete reload or reboot. A test module 106 may dynamically reinstall its test module interface at any time without rebooting computer system 100. A test module 106 may dynamically reinstall its test module interface in response to a device change associated with the test module 106. For example, a device such as device 140a, 140b, or 140c may be added to, removed from, or substituted for another device in computer system 100, or another test module may expose an additional device. In response, a test module 106 associated with the changed device may be loaded or unloaded, as appropriate, and may cause a test module interface to be installed, deleted, or reinstalled, as appropriate, without the need to reboot computer system 100. Where the test module interface includes an EFI protocol, the process of installing, deleting, or reinstalling the test module interface may also include notifying an executive of the change. Accordingly, the

executive can cause a test configuration to be created or recreated where the test module interface is installed or reinstalled and can stop using a test module interface, and hence the test module 106, where the test module interface is deleted.

To allow a test module 106 to be safely unloaded from memory, a registration module is used to register uses of the test module 106 by other programs. By registering uses of the test module 106, the test module 106 can insure that no other program attempts to access information associated with the test module 106 after it unloads. An example of a registration table used by a registration module is described below in Fig. 4a. In response to a test module 106 seeking to unload, the registration module is accessed to determine what other programs are registered as using the test module 106. Each of these programs is notified to stop using the test module 106. After each of the programs responds that it has stopped using the test module 106, the test module 106 can cause itself to unload. If a program responds that it is still using the test module 106, then the test module 106 can cancel the unload. The test module 106 may seek to unload at a later time.

In one particular embodiment, a test module 106 causes a QUITUSING function to be called for each program that is registered in the registration module as using the test module 106. Each QUITUSING function returns either an OK signal indicating that the test module 106 can unload or a DEFER signal indicating that the test module 106 should not unload. If each QUITUSING function returns an OK signal, then the test module 106 causes itself to be unloaded. If one or more of the QUITUSING functions returns a DEFER signal, then the test module causes the unloading to be canceled and does not unload. The test module 106 may attempt to unload at a later time.

The registration module can also be used in the process of reinstalling a test module interface. In response to a test module 106 seeking to reinstall its interface, possibly in response to a condition noted above, the test module 106 determines whether a program dependency exists using the registration module. In particular, the test module 106 determines whether another program is registered as using the test module 106. Each

program that is registered as using the test module 106 is notified to stop using it. After each of the programs responds that it has stopped using the test module 106, the test module 106 can cause its test module interface to be reinstalled. If a program responds that it is still using the test module 106, then the test module 106 can cancel the reinstall. The test module 106 may seek to reinstall at a later time. In one particular embodiment, the QUITUSING function described above is used to handle these program dependencies.

The embodiment of computer system 100 shown in Fig. 1 is shown by way of example. Other computer systems may include other numbers or types of processors, devices, chipsets, and/or memories and these components may be connected in ways other than the ways shown in Fig. 1. Devices 140a, 140b, and 140c may each be any device suited for use in a computer system including a hard disk drive, a CDROM drive, a graphics subsystem, an audio subsystem, a printer, a scanner, or an input device such as a keyboard or mouse. Memory 130 may be any memory or combination of memory elements suited for use in a computer system including any type of ROM, RAM, hard disk drive, or other type of magnetic or optical storage device.

In one particular embodiment, test modules 106 are stored onto a hard disk drive on computer system 100 and are loaded to and unloaded from memory 130 in order to be executed. In this embodiment, test module interfaces are stored in memory 130.

Fig. 2 is a diagram illustrating layers of software in a diagnostic architecture. Fig. 2 shows a high level diagram of the diagnostic architecture described in Fig. 1. In Fig. 2, a user interface layer 210, an executive module layer 220, a test module layer 230, and a driver layer 240 are shown. Each software layer 210, 220, 230, and 240 allows other layers to interact with it using an interface such as the test module interface described above.

Test modules and device drivers are loaded into a memory and expose interfaces. In some instances, a test module is included in a device driver. The test module interface provides a way for an executive to access information in the test module. Similarly, an executive exposes an executive module interface to allow a test module and a user interface

to access information in the executive module, and the user interface exposes a user interface interface to allow an executive module to access information in the user interface. Each interface may utilize an EFI protocol as noted above. The executive module and/or the user interface may cause the test modules to be loaded into or unloaded from a memory.

Fig. 3 is a diagram illustrating an example of the use of test module interfaces in a diagnostic architecture. In Fig. 3, a program 300 that includes a user interface 310 and an executive module 320 is shown. User interface 310 exposes user interface interface (“UIIF”) 312 and executive module 320 exposes executive module interface (“EMIF”) 324. Fig. 3 also includes test modules 330a, 330b, and 330c and a registration module 340. Test modules 330a, 330b, and 330c expose test module interfaces (“TMIF”) 332a, 332b, and 332c, respectively, and registration module 340 exposes TMIF 332d. Test module 330c also exposes additional interfaces 334a and 334b, and registration module 340 also exposes interface 342. The various interfaces 312, 324, 332a, 332b, 332c, 332d, 334a, 334b, and 342 may be exposed in response to each interface’s associated program being loaded.

The arrows in Fig. 3 represent pointers that are stored by each program to interfaces. For example, user interface 310 stores a pointer 314 to EMIF 324, and executive module 320 stores a pointer 326 to UIIF 326. Executive module 320 detects the test module interfaces using EFI and stores pointers 328a, 328b, and 328c to TMIFs 330a, 330b, and 330c, respectively. Executive module 320 also detects TMIF 327 and interface 342 of registration module 340 and stores pointers 327 and 329, respectively. As described above, test modules 330a, 330b, and 330c may cause their respective TMIFs 332a, 332b, and 332c to be installed in response to the presence or absence of certain conditions.

After executive module 320 detects TMIFs 332a, 332b, and 332c, it calls functions identified by TMIFs 332a, 332b, and 332c to cause test configurations (not shown in Fig. 3) to be created for test modules 330a, 330b, and 330c, respectively. Executive module 320 can then pass information from the test configurations to user interface 310 and call test programs identified by the test configurations for execution.

Executive module 320 also registers its use of test modules 330a, 330b, and 330c with registration module 340. Registration module 340 creates a table in a memory to maintain a list of programs that are using other programs such as test modules. Fig. 4a is a diagram illustrating an embodiment of a registration table 400. Registration table 400 includes entries, such as entry 408, that each include a module ID 402, a pointer 404, and a function indicator 406. Module ID 402 identifies the program registering its use of another program. Pointer 404 identifies the program being used by the program identified by module ID 402. Function indicator 406 identifies a function that may be called to request that the program identified by module ID 402 stop using the program associated with pointer 404. Registration table 400 includes one entry for each registered use of a program. Other embodiments of registration tables may store other types of information.

The operation of registration table 400 is seen by way of an example. Figs. 4b, 4c, and 4d are diagrams illustrating registration table 400 as it might appear a various times in the example of Fig. 3. In response to detecting TMIFs 332a, 332b, and 332c, executive module 320 registers its use of test modules 330a, 330b, and 330c, respectively, using registration module 340. Registration module 340 creates an entry for each of these uses—entries 422, 424, and 426 in registration table 400 shown in Fig. 4b. As indicated in Fig. 4b, entry 422 includes a module ID (executive module 320 ID), a pointer (TMIF pointer 332a), and a function indicator (FQUITUSING). Similarly, entry 424 includes a module ID (executive module 320 ID), a pointer (TMIF pointer 332b), and a function indicator (FQUITUSING), and entry 426 includes a module ID (executive module 320 ID), a pointer (TMIF pointer 332c), and a function indicator (FQUITUSING). Executive module 320 ID may be executive module 320's globally unique identifier ("GUID") or another unique number assigned by EFI in each entry 422, 424, and 426. The pointers in each entry may be pointers to TMIF 332a, TMIF 332b, and TMIF 332c, respectively. The functions in each entry may refer to a function in executive module 320 that may be called to request that

executive module 320 stop using the test modules 330a, 330b, and 330c associated with TMIF 332a pointer, TMIF 332b pointer, and TMIF 332c pointer, respectively.

In Fig. 3, pointer 338a indicates that test module 330a is using test module 330c via interface 334a. Similarly, pointer 338b indicates that test module 330b is using test module 330c via interface 334b. Accordingly, test modules 330a and 330b register their respective uses with registration module 340, and registration module 340 creates entries 428 and 430 as shown in Fig. 4b. Test module 330a ID may be test module 330a's GUID assigned by EFI, and test module 330b ID may be test modules 330b's GUID assigned by EFI.

At some point in operation, test module 330c or another test module may attempt to cause itself to be unloaded from memory. After being unloaded from memory, other programs will not be able to access information and test programs in test module 330c and any attempt by another program to do so may cause a system to become unstable. In order to unload from memory, test module 330c uses registration module 340 to notify other programs that are using it that it is seeking to unload. In particular, registration module 340 identifies the programs that are registered as using test module 330c using registration table 400 and calls the appropriate function to notify the programs to stop using test module 330c. As indicated by shaded entries 426, 428, and 430 in Fig. 4c, executive module 320, test module 330a, and test module 330b are registered as using test module 330c. Registration module 340 calls the appropriate function, in this case FQUITUSING, for executive module 320, test module 330a, and test module 330b. After executive module 320, test module 330a, and test module 330b respond to registration module 340 with a signal indicating that they have stopped using test module 330c, registration module 340 notifies test module 330c that it may unload. Registration module 340 also deletes the entries in registration table 400 associated with test module 330c to result in registration table 400 shown in Fig. 4d. Test module 330c can then cause itself to be unloaded. If either executive module 320, test module 330a, or test module 330b respond to registration module with a signal indicating that test module 330c is still being used, then registration module 340 notifies test module



330c that it may not unload. Test module 330c can then cancel its unload and may attempt to unload at a later time.

At some point in operation, test module 330c or another test module may attempt to reinstall TMIF 332c as indicated by dashed arrow 350. Test module 330c and registration module 340 may follow an analogous notification process just described before test module 330c reinstalls TMIF 332c. After executive module 320, test module 330a, and test module 330b respond to registration module 340 with a signal indicating that they have stopped using test module 330c, registration module 340 notifies test module 330c that it may reinstall TMIF 332c. Registration module 340 also deletes the entries in registration table 400 associated with test module 330c to result in registration table 400 shown in Fig. 4d. Test module 330c can then delete and reinstall TMIF 332c. If either executive module 320, test module 330a, or test module 330b respond to registration module with a signal indicating that test module 330c is still being used, then registration module 340 notifies test module 330c that it may not reinstall TMIF 332c. Test module 330c can then cancel the reinstall and may attempt to reinstall TMIF 332c at a later time.

Fig. 5a is a flowchart illustrating an embodiment of a method for using test module interfaces in a diagnostic architecture. In Fig. 5a, test modules are loaded as indicated in step 502. Test module interfaces are detected as indicated in step 504. Uses of test modules are registered as indicated in step 506. A configuration function is performed for each test module as indicated in step 508. A test is performed using a test module as indicated in step 510. Uses of test modules are unregistered as indicated in step 512. Test modules are unloaded as indicated in step 514.

Fig. 5b is a flowchart illustrating an embodiment of a method for unloading a test module in a diagnostic architecture. In Fig. 5b, the unloading of a test module is initiated as indicated in step 520. A QUITUSING function is called for each dependent program as indicated in step 522. A determination is made as to whether a QUITUSING function returns a “defer” response as indicated in step 524. If a QUITUSING function returns a “defer”

response, then the test module is not unloaded as indicated in step 528. If no QUITUSING function returns a “defer” response, then the test module is unloaded as indicated in step 526.

Fig. 5c is a flowchart illustrating an embodiment of a method for reinstalling a test module interface in a diagnostic architecture. In Fig. 5c, a determination is made as to whether a reinstall has been initiated as indicated in step 530. The chart remains at step 530 until a reinstall has been initiated. Once a reinstall has been initiated, then a determination is made as to whether any program dependencies exist as indicated in step 532. If a program dependency does not exist, then the interface is reinstalled as indicated in step 538.

If a program dependency exists, then a QUITUSING function is called for each dependent program as indicated in step 534. A determination is made as to whether any QUITUSING function returns a “defer” response as indicated in step 536. If no QUITUSING function returns a “defer” response, then the interface is reinstalled as indicated in step 538. If a QUITUSING function returns a “defer” response, then the interface is not reinstalled as indicated in block 540.

Fig. 6 is a diagram illustrating an example of a test configuration 650. Test configuration 650 includes test module information 600, device information 604, device information 624, test information 608, test information 620, parameter information 612, parameter information 616, parameter information 628, and parameter information 632. Test module information 600 includes a pointer 602 to device information 604. Device information 604 includes a pointer 606 to test information 608 and a pointer 622 to device information 624. Test information 608 includes a pointer 618 to test information 620 and a pointer 610 to parameter information 612. Parameter information 612 includes a pointer 614 to parameter information 616. Device information 624 includes a pointer 626 to parameter information 628. Parameter information 628 includes a pointer 630 to parameter information 632.

Test module information 600 includes module specific information such as a module name, a vendor name, help text, and a version, a pointer to a device list such as pointer 602, and a pointer (not shown) to its associated test module interface.

Device information 604 and device information 624 each include device specific information such as a device name, a mnemonic, help text, and configuration text, a pointer (not shown) to a test module information such as test module information 600, a next pointer to a next device such as pointer 622, a pointer to a list of tests for this device such as pointer 606, and a pointer to a set of parameters for this device such as pointer 626.

Test information 608 and test information 620 each include test specific information such as a test name, a mnemonic, help test, a test number, and flags for controlling the behavior of a test function, a pointer (not shown) to a test program entry point, pointers to a list of parameters that are specific to a test such as pointer 610, a pointer to a next test such as pointer 618, and a pointer (not shown) back to a device.

Parameter information 612, parameter information 616, parameter information 628, and parameter information 632 each include parameter specific information such as a parameter name, a mnemonic, help text, flags for controlling the display of the parameter, and a type of the parameter, a pointer to a next parameter such as pointer 630, and parameter data such as a pointer (not shown) to a list of choices for the parameter.

As described above, an executive causes test configuration 650 of a test module to be created and stored in a memory using a test module interface of the test module. In the example shown in Fig. 6, the executive receives a pointer 660 to indicate a location of test configuration 650 in memory. The executive then parses the test configuration to detect test programs and other information available in the test module. The executive can provide information from test configuration 650 to a user interface and the user interface can present the information to a user in any appropriate form such as a test menu or a graphical point and click interface.

As can be seen, a principal advantage of this embodiment is that it allows a test module to be used in a build-to-order manufacturing environment. The test module is loaded onto a build-to-order computer system. A test module interface associated with the test module provides a user interface or other program with the ability to select test routines or other information in the test module. The test module interface may be created using the capabilities of an interface between an operating system and platform firmware. In addition, the use of an interface between an operating system and platform firmware allows the test module interface to be registered with a registration module and to be dynamically reinstalled. The registration module provides test modules with the ability to register uses of a particular test module to ensure that that particular test module does not unload prematurely.

Although illustrative embodiments have been shown and described, a wide range of modification, change and substitution is contemplated in the foregoing disclosure and in some instances, some features of the embodiment may be employed without a corresponding use of other features. Accordingly, it is appropriate that the appended claims be construed broadly and in a manner consistent with the scope of the embodiments disclosed herein.

Claims

What is claimed is:

1. A method performed by a computer system comprising:
  - detecting a test module interface associated with a test module; and
  - calling a function identified by the test module interface to cause a first test configuration of the test module to be created.
2. The method of claim 1, further comprising:
  - detecting a test routine associated with the test module using the first test configuration; and
  - causing the test routine to be executed.
3. The method of claim 1, further comprising:
  - loading the test module;
  - detecting a device associated with the test module; and
  - in response to detecting the device, creating the test module interface.
4. The method of claim 3, further comprising:
  - detecting a change associated with the device; and
  - in response to detecting the change, reinstalling the test module interface associated with the test module.

- 1 5. The method of claim 4, further comprising:  
2 in response to detecting the change,  
3 calling the function to cause a second test configuration of the test module to be  
4 created.
- 1 6. The method of claim 1, further comprising:  
2 registering a use of the test module by a program.
- 1 7. The method of claim 6, further comprising:  
2 unloading the test module; and  
3 informing the program of the unloading prior to unloading the test module.
- 1 8. The method of claim 7, further comprising:  
2 conveying a defer signal from the program to the test module; and  
3 in response to the defer signal, canceling the unloading of the test module.
- 1 9. A computer system comprising:  
2 a processor; and  
3 a memory coupled to the processor and including a program and a test module;  
4 the program being executable by the processor to:  
5 detect a test module interface associated with a test module; and  
6 call a function identified by the test module interface to cause a first test  
7 configuration of the test module to be created.
- 1 10. The computer system of claim 9, wherein the program is executable to:  
2 detect a test routine associated with the test module using the first test configuration;  
3 and  
4 cause the test routine to be executed.

- 1 11. The computer system of claim 9, wherein the test module is executable to:  
2 detect a device associated with the test module; and  
3 in response to detecting the device, create the test module interface.
- 1 12. The computer system of claim 11, wherein the test module is executable to:  
2 detect a change associated with the device; and  
3 in response to detecting the change, reinstall the test module interface associated with  
4 the test module.
- 1 13. The computer system of claim 12, wherein the program is executable to:  
2 call the function identified by the test module interface to cause a second test  
3 configuration of the test module to be created.
- 1 14. The computer system of claim 9, wherein the program is executable to:  
2 cause a use of the test module by the program to be registered.
- 1 15. The computer system of claim 14, wherein the test module is executable to:  
2 cause the test module to be unloaded; and  
3 cause the program to be notified prior to unloading the test module.
- 1 16. The computer system of claim 15, wherein the program is executable to:  
2 convey a defer signal from to the test module; and  
3 wherein the test module is executable to:  
4 in response to the defer signal, cause the unloading of the test module to be  
5 canceled.

1 17. A computer system comprising:  
2 a processor; and  
3 a memory coupled to the processor and including a first program and a second  
4 program that includes an interface;  
5 the first program being executable by the processor to:  
6 call a first function identified by the interface, the first function being  
7 executable by the processor to register a use of the second program by  
8 the first program.

1 18. The computer system of claim 17, wherein the first function is executable by the  
2 processor to cause an entry associated with the first program to be stored in the memory.

1 19. The computer system of claim 18, wherein the entry includes a first identifier  
2 associated with the first program, a second identifier associated with the interface, and a third  
3 identifier associated with a second function.

1 20. The computer system of claim 17, wherein the second program is executable by the  
2 processor to:  
3 cause the second program to be unloaded from the memory; and  
4 cause the first program to be notified that the second program seeks to be unloaded.

1 21. The computer system of claim 20, wherein the first program is executable by the  
2 processor to:  
3 convey a defer signal to the second program in response to being notified that the  
4 second program seeks to be unloaded; and  
5 wherein the second program is executable by the processor to:  
6 cause the second program not to be unloaded in response to receiving the defer signal.



22. The computer system of claim 17, wherein the first program is executable by the processor to:

call a second function identified by the interface, the second function being executable by the processor to cause a configuration of the second program to be created.

[illegible]

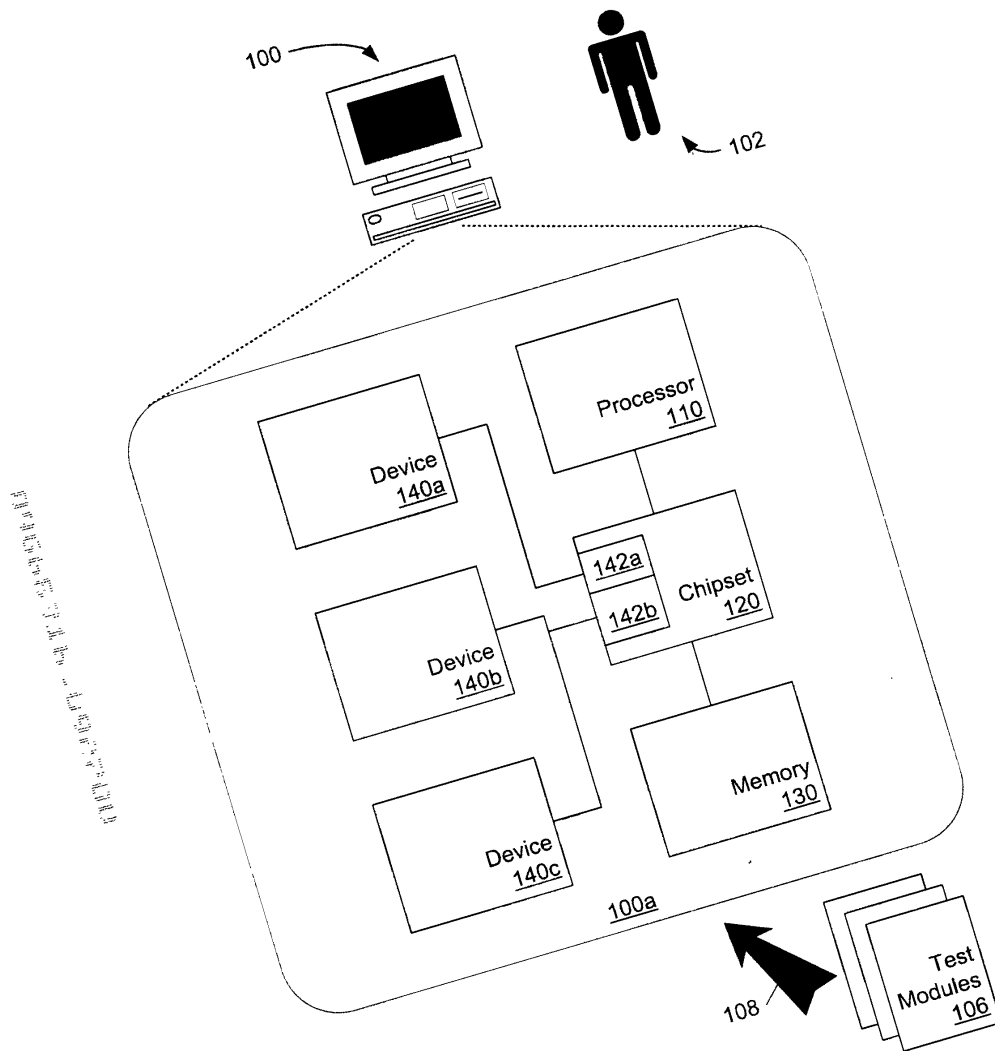
# A DIAGNOSTIC ARCHITECTURE FOR USE WITH AN INTERFACE BETWEEN AN OPERATING SYSTEM AND PLATFORM FIRMWARE

### Abstract of Disclosure

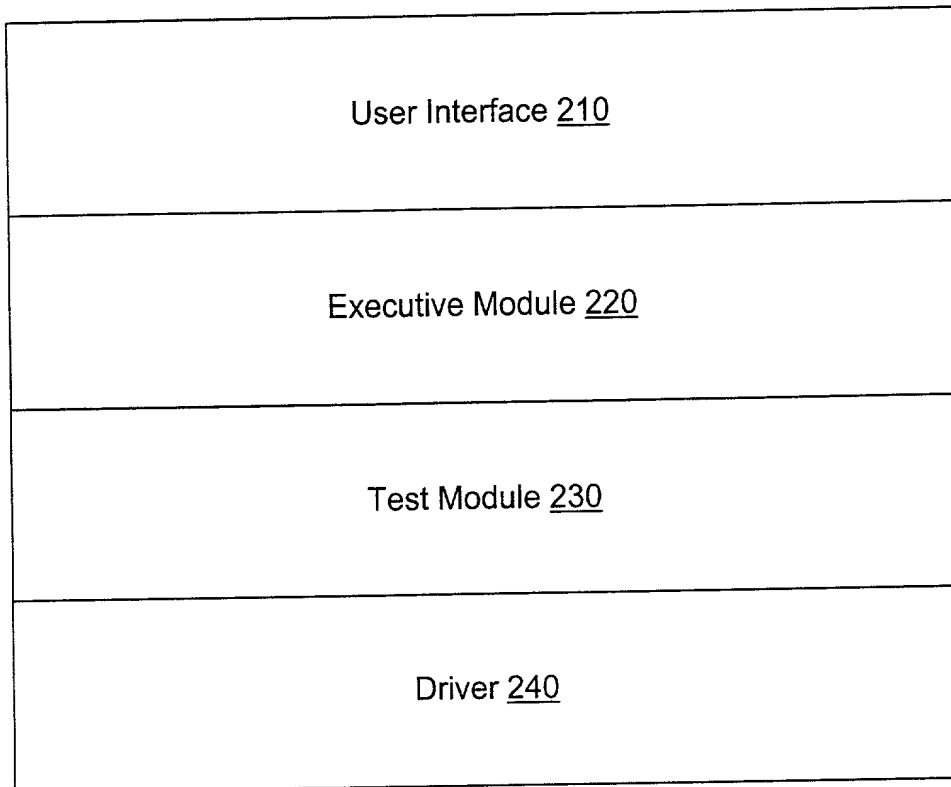
5

A method performed by a computer system is provided. The method includes detecting a test module interface associated with a test module and calling a function identified by the test module interface to cause a test configuration of the test module to be created.

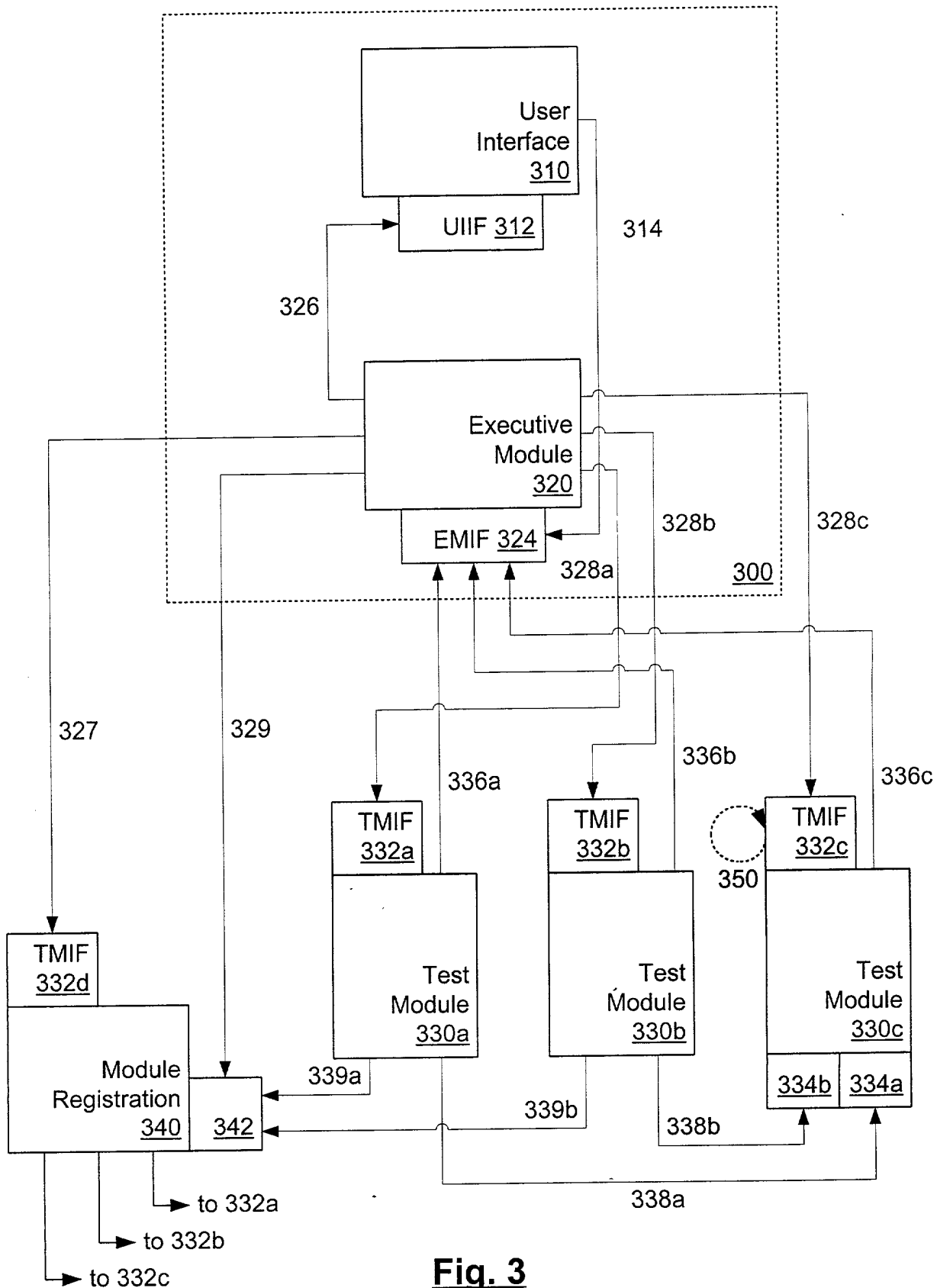
a-101135v1



**Fig. 1**



**Fig. 2**

**Fig. 3**

408

**Fig. 4a**

430

**Fig. 4b**

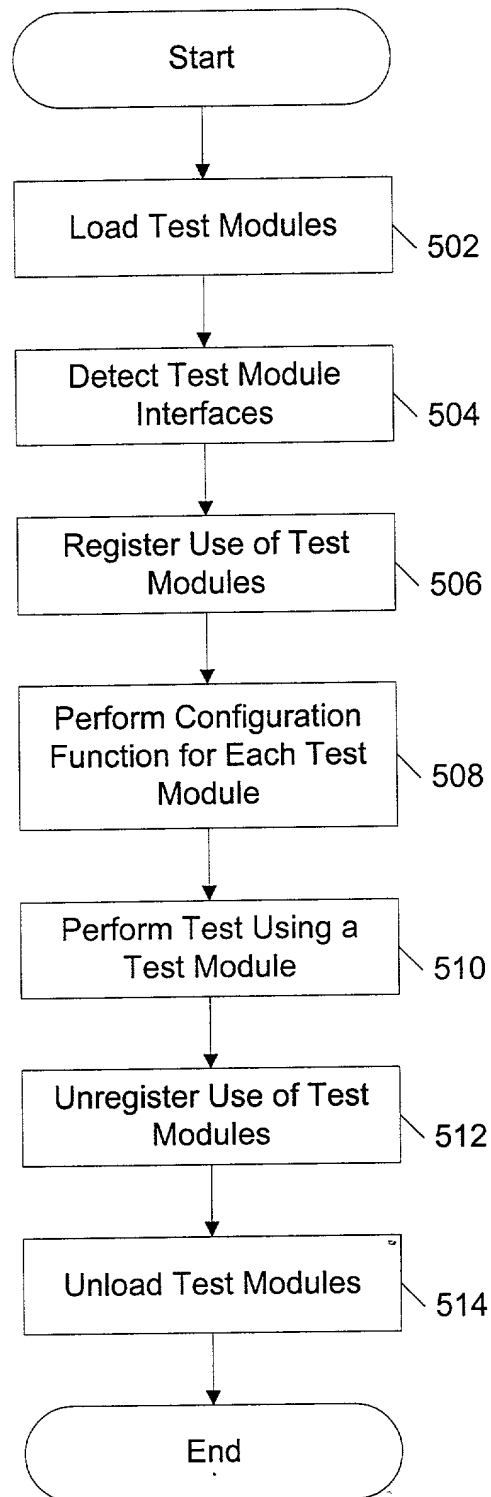
Executive Module 320 ID	TMIF 332a Pointer	fQuitUsing	422
Executive Module 320 ID	TMIF 332b Pointer	fQuitUsing	424
Executive Module 320 ID	TMIF 332c Pointer	fQuitUsing	426
Test Module 330a ID	TMIF 332c Pointer	fQuitUsing	428
Test Module 330b ID	TMIF 332c Pointer	fQuitUsing	430

400  **Fig. 4c**

Executive Module 320 ID	TMIF 332a Pointer	fQuitUsing	422
Executive Module 320 ID	TMIF 332b Pointer	fQuitUsing	424

400  **Fig. 4d**

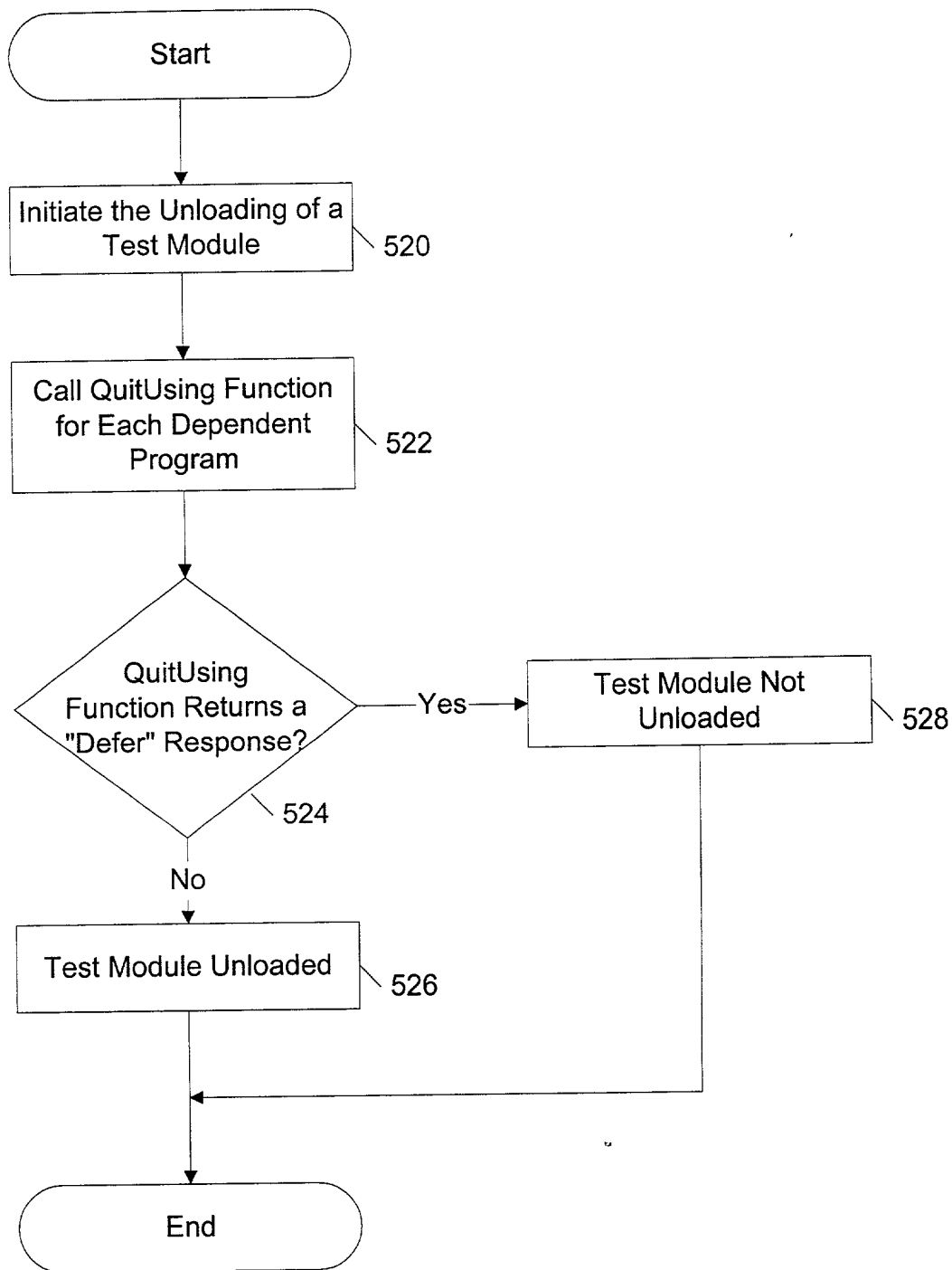
0966316-096600



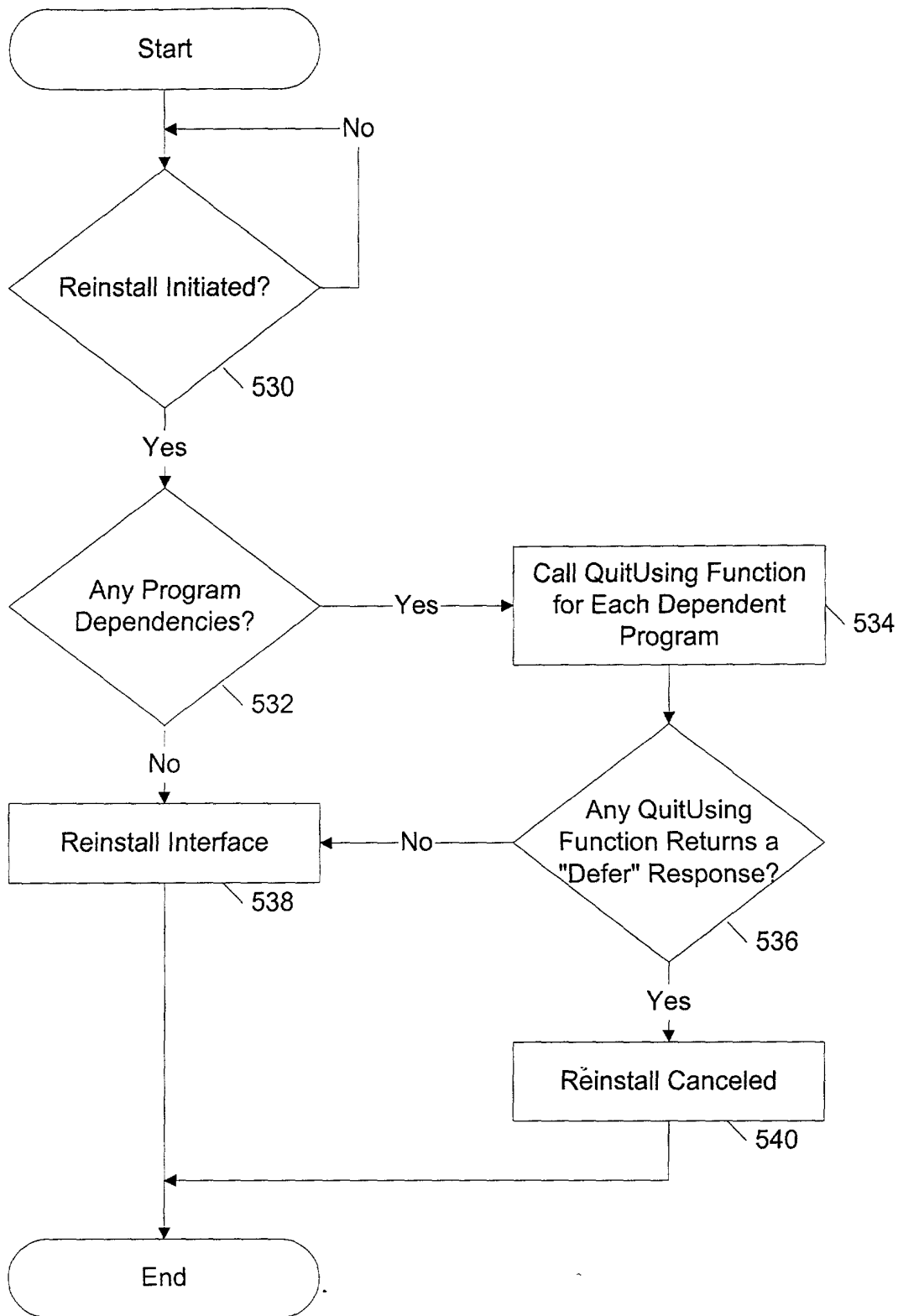
**Fig. 5a**

002260 " 9T83960

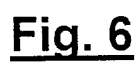




**Fig. 5b**



**Fig. 5c**



**Fig. 6**

DECLARATION AND POWER OF ATTORNEY FOR  
PATENT APPLICATION

As below named inventors, we hereby declare that:

Our residence, post office address and citizenship are as stated below next to our names;

We believe we are the original, first and joint inventors of the subject matter which is claimed and for which a patent is sought on the invention entitled

**A DIAGNOSTIC ARCHITECTURE FOR USE WITH AN INTERFACE  
BETWEEN AN OPERATING SYSTEM AND PLATFORM FIRMWARE**

the specification of which: (check one)

XXX is attached hereto.

\_\_\_\_\_ was filed on \_\_\_\_\_  
under Attorney's Docket Number \_\_\_\_\_  
as Application Serial No. \_\_\_\_\_  
and was amended on \_\_\_\_\_ (if applicable).

We hereby state that we have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

We acknowledge the duty to disclose information which is material to the patentability of this application in accordance with 37 CFR 1.56.

We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 USC 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

002239 "AT" 09200

Jeffrey M. Becker	Reg. No. 35,442	Mark P. Kahler	Reg. No. 29,178
James R. Bell	Reg. No. 26,528	Warren B. Kice	Reg. No. 22,732
Michael S Bush	Reg. No. 31,745	Christopher P. Kosh	Reg. No. 42,760
Randall E. Colson	Reg. No. 40,566	Michael J. Balconi-Lamica	Reg. No. 34,291
Michael A. Davis, Jr.	Reg. No. 35,488	Todd Mattingly	Reg. No. 40,298
Ruben C. DeLeon	Reg. No. 37,812	David L. McCombs	Reg. No. 32,271
Anthony Peterman	Reg. No. 38,270	David M. O'Dell	Reg. No. 42,044
Henry Garrana	Reg. No. 27,887	Phillip B. Philbin	Reg. No. 35,979
Timothy Headley	Reg. No. 31,765	Diana L. Roberts	Reg. No. 36,654
Rita M. Irani	Reg. No. 31,028	Brandi W. Sarfatis	Reg. No. 37,713

POST OFFICE ADDRESS: P.O. Box 181156, Austin, Texas 78718